Sé diferente, intégrate...

Mca073

Manual RUBY on RAILS UN FRAMEWORK PARA DESARROLLAR APLICACIONES WEB RUBY

Fecha: 01 /06 / 2013

AUTOR: ORLANDO GUTIERREZ

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

INDICE

Pag. 3 L1 Introducción a Ruby : Lenguaje de Programación, Ventajas y Desventajas
Pag. 4 L2 Datos: Números, Strings
Pag. 6 L3 Variables
Pag. 7 L4 Entrada de datos
Pag. 7 L5 Rangos
Pag. 8 L6 Arreglos
Pag. 9 L7 Instrucciones Secuenciales
Pag. 9 L8 Instrucciones Condicionales
Pag. 10 L9 Instrucciones Repetitivas
Pag. 11 L10 Programación Orientada a Objetos
Pag. 12 L11 Manejo de archivos
Pag. 13 L12 Módulos
Pag. 14 L13 Excepciones
Pag. 15 L14 Desarrollo de Aplicaciones WEB
Pag. 18 L15 Desarrollo de aplicaciones en múltiples capas
Pag. 19 L16 Framewoks de desarrollo
Pag. 20 L17 ¿Qué es Rails?
Pag. 21 L18 Instalación de Rails: Windows, Linux
Pag. 21 L19 Framework Ruby On Rails: Modelo, Vista, Controlador
Pag. 23 L20 Estructura directorios Rails
Pag. 24 L21 Desarrollo aplicaciones en Rails
Pag. 25 L22 Configuración de Bases de Datos para aplicaciones Rails
Pag. 26 L23 Capa de Modelo en Rails
Pag. 32 L24 Capa de Vista en Rails
Pag. 37 L25 Capa de Controlador en Rails
Pag. 38 L26 Layouts en Rails
Pag. 41 L27 Desarrollo de Prototipos en Rails
Pag. 41 L28 Combinando Rails con AJAX
Pag. 42 L29 Descarga de archivos en Rails
Pag. 43 L30 Envío de emails en Rails

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

L1 Introducción a Ruby : Lenguaje de Programación, Ventajas y Desventajas

Características

Ruby es un lenguaje de “scripts” para programar orientado por objetos.

Es interpretado:

• Permite realizar directamente llamadas al sistema de operación

• Ofrece operaciones de alto nivel sobre cadenas de caracteres y expresiones regulares

• Retroalimentación inmediata durante el proceso de desarrollo

Es eficiente:

• Son innecesarias las declaraciones de variables

• Las variables no tienen tipo

• La sintaxis es simple y consistente

• La gestión de la memoria es automática

Es orientado por objetos:

• Todo es un objeto

• Facilidades de Clases, herencia, métodos, entre otros

• Métodos singleton

• Mixins por módulos

• Iteradores y cierres

También ofrece:

• Enteros de precisión múltiple

• Modelo de procesamiento de excepciones

• Carga dinámica

• Hilos

Historia

El lenguaje fue creado por Yukihiro "Matz" Matsumoto, quien empezó a trabajar en Ruby el 24 de febrero de 1993, y lo presentó al público en el año 1995. En el círculo de amigos de Matsumoto se le puso el nombre de "Ruby" (en español rubí) como broma aludiendo al lenguaje de programación "Perl" (perla).

Desventajas de Ruby

A continuación se enumeran los criterios comparándose con otros lenguajes:

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

- El rendimiento comparable a Perl o Python, pero lejos de C o C++
- No existen muchas frameworks desarrollados en Ruby; Ruby onRails (http://www.rubyonrails.com/) es la excepción.
- No existe un framework de GUI multi-plataforma ampliamente aceptada. RAA – Ruby Application Archive (http://raa.ruby-lang.org/) no es tan grande como CPAN – Comprehensive Perl Active Network (http://www.cpan.org)

Los desarrolladores de Ruby aun critican los siguientes aspectos:

- Documentación en progreso
- No es un lenguaje demasiado conocido en países occidentales
- El diseño se orienta más hacia Linux que Windows
- No tiene un buen soporte de Unicode

L2 Datos: Números, Strings

Ruby maneja tanto strings como datos numéricos.

Datos numéricos

Los datos numéricos pueden ser números enteros y números reales.

Cualquier operación involucrando un número real procesa números reales, solo las operaciones conteniendo únicamente números enteros procesan números enteros.

Datos Strings o cadenas

Las cadenas pueden estar entre comillas dobles ("...") o comillas simples ('...').

Las comillas simples y dobles tienen efectos diferentes. Una cadena de comillas dobles permite la presencia de **caracteres de escape precedidos por un backslash** y la **expresión de evaluación #{ }**. Una cadena de comillas simples no realiza esta evaluación. Ejemplos:

```
ruby> print "a\nb\nc","\n"
a
b
c
nil
ruby> print 'a\nb\nc',"\n"
a\nb\nc
nil
ruby> "\n"
"\n"
ruby> '\n'
```

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

```
"\\n"

ruby> "\001"

"\001"

ruby> '\001'

"\\001"

ruby> "abcd #{5*3} efg"

"abcd 15 efg"

ruby> var = " abc "

" abc "

ruby> "1234#{var}5678"

"1234 abc 5678"
```

Ruby ofrece operadores sobrecargados para el manejo de las cadenas:

Concatenación (**operador +**):

```
ruby> word = "fo" + "o"

"foo"
```

Repetición (**operador ***):

```
ruby> word = word * 2

"foofoo"
```

Extracción de caracteres (**operador []** , nota: los caracteres en Ruby son enteros):

```
ruby> word[0]

102 # 102 es el código ASCII de 'f'

ruby> word[-1] # última posición se accede con índice -1

111 # 111 es el código ASCII de 'o'
```

(Los índices negativos indican desplazamientos desde el final de la cadena, en vez del comienzo).

Extracción de subcadenas (**operador [,]**):

```
ruby> herb = "parsley"

"parsley"

ruby> herb[0,1]

"p"

ruby> herb[-2,2]
```

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

"ey"

ruby> herb[0..3]

"pars"

ruby> herb[-5..-2]

"rsle"

Verificación de la igualdad (**operador ==**) :

ruby> "foo" == "foo"

true

ruby> "foo" == "bar"

false

L3 Variables

Ruby tiene tres clases de variables, una clase de constante y exactamente dos pseudo-variables. Las variables y las constantes no tienen tipo. En Ruby, las variables no se declaran.

El tipo de variable se determina por el prefijo de la misma, en la siguiente tabla se muestra:

PREFIJO	TIPO DE VARIABLE
$	Variable Global
@	Variable Instancia
[a-z] ó _	Variable Local
[A-Z]	Constante

Existen dos pseudovariables definidas en Ruby. La variable **self**, referenciando al objeto actual en ejecución y la variable **nil ,** es el valor que toman las variables no inicializadas.

Variables globales:

Una variable global tiene un nombre que comienza con **$**. Se puede utilizar en cualquier parte de un programa. Antes de inicializarse, una variable global tiene el valor especial **nil.**

En la siguiente tabla se muestran las variables globales internamente almacenadas por Ruby.

VARIABLE	VALOR
$!	Último mensaje de error
$@	Posición del error
$_	Última cadena leída con **gets**
$.	Último número de línea leído por el interprete
$&	Última cadena que ha coincidido con una expresión Regular
$~	Última cadena que ha coincidido con una expresión regular como array de subexpresiones
$*n*	La n-ésima subexpresión regular de la última coincidencia (igual que $~[n])
$=	flag para tratar igual las mayúsculas y minúsculas
$/	Separador de registros de entrada
$\	Separador de registros de salida
$0	El nombre del fichero del guión Ruby
$*	El comando de la línea de argumentos

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

$$	El número de identificación del proceso del intérprete Ruby
$?	Estado de retorno del último proceso hijo ejecutado

Variables de instancia:

Se refiere a los campos de un objeto o instancia de una clase.

Una variable de instancia tiene un nombre empezando con @ y su ámbito está limitado al objeto referenciado por self. Dos objetos diferentes, aún cuando pertenezcan a la misma clase, pueden tener valores diferentes en sus variables de instancia. Desde el exterior del objeto, las variables de instancia, no se pueden alterar e incluso, no se pueden observar (en Ruby las variables de instancia nunca son públicas) a excepción de los métodos proporcionados explícitamente por el programador. Como con las variables globales, las variables de instancia tienen el valor nil antes de ser inicializadas. Las variables de instancia en Ruby no necesitan declararse. Esto da lugar a una estructura flexible de los objetos. Cada variable de instancia se añade dinámicamente al objeto la primera vez que se la referencia.

Variables locales:

Una variable local tiene un nombre que empieza con una letra minúscula o con el carácter de subrayado (_). Las variables locales no tienen, a diferencia de las variables globales y las variables de instancia, el valor nil antes de la inicialización. La primera asignación realizada sobre una variable local actúa como una declaración. Si se referencia a una variable local no inicializada, el intérprete de Ruby piensa se trata de una llamada a un método con ese nombre y se produce un mensaje de error.

El operador **define?**, indica si un identificador se encuentra definido. Si lo está, devuelve una descripción del mismo, en caso contrario, devuelve **nil**

L4 Entrada de datos

En Ruby, **ARGV** es un arreglo conteniendo la especificación en la línea de comandos.

La función **STDIN.gets**, almacena en una cadena, el valor introducido en el teclado.

Ruby provee un conjunto de métodos relacionados con la entrada y la salida implementados en el módulo Kernel. Todos estos métodos derivan de la clase IO. La clase The class IO provee los métodos básicos como **read**, **write**, **gets, puts, readline, getc**, y **printf**.

La instrucción **gets** puede ser utilizada para tomar cualquier entrada del usuario desde la pantalla estándar de datos, conocida como STDIN

A continuación se muestra un ejemplo

```
puts "Introduzca un valor :"

val = gets

puts val
```

L5 Rangos

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

Los rangos en Ruby se indican con la expresión regular **[]**. Con el operador [], se especifica un rango; ejemplo [a-z] representa una letra en el rango desde la a hasta la z.

Existen funciones para el manejo de rangos, por ejemplo **rand(3)** devuelve un número aleatorio dentro del rango de 0 a 2.

2..5 es una expresión que representa un rango entre 2 y 5 inclusive.

L6 Arreglos

Se puede crear un arreglo listando elementos entre **corchetes ([ ])** y separándolos por comas. Los arreglos en Ruby pueden almacenar objetos de diferentes tipos.

```
ruby> ar = [1, 2, "3"]

[1, 2, "3"]
```

Los arreglos se pueden concatenar **(+)** y repetir **(*),** igual que las cadenas.

```
ruby> ar + ["foo", "bar"]

[1, 2, "3", "foo", "bar"]

ruby> ar * 2

[1, 2, "3", 1, 2, "3"]
```

Se pueden utilizar índices numéricos para acceder a cualquier parte del arreglo.

```
ruby> ar[0]

1

ruby> ar[0,2]

[1, 2]

ruby> ar[-2]

2

ruby> ar[-2,2]

[2, "3"]

ruby> ar[-2..-1]

[2, "3"]
```

(Los índices negativos indican se empieza a contar desde el final del arreglo, en lugar desde el principio).

Los arreglos se pueden convertir en y obtener de cadenas utilizando las funciones **join** y **split** respectivamente:

```
ruby> str = ar.join(':')

"1:2:3"

ruby> ar.split(':')
```

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

L7 Instrucciones Secuenciales

Las instrucciones en Ruby se separan por salto de línea. No existe sintaxis de separación de instrucciones; en el caso de las instrucciones conteniendo bloques, las mismas contienen la instrucción final **end**.

Si los bloques de instrucciones contienen una sola instrucción, la palabra clave **end** puede omitirse.

L8 Instrucciones Condicionales

Simple

```
if <condición>
    …. <bloque instrucciones separadas por líneas>
end
```

Si no

```
if <condición>
    …. <bloque instrucciones separadas por líneas>
else
    …. <bloque instrucciones separadas por líneas>
end
```

Switch

```
case <variable>
  when <condición>
    …. <bloque instrucciones separadas por líneas>
  when <condición>
    …. <bloque instrucciones separadas por líneas>
end
```

Se utiliza la sentencia case para comprobar una secuencia de condiciones. La sintaxis es similar al switch de C y Java pero permite incorporar rangos y expresiones regulares, como es mostrado en el ejemplo siguiente.

```
ruby> i=8
8
ruby> case i
```

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

```
ruby| when 1, 2..5

ruby| print "1..5\n"

ruby| when 6..10

ruby| print "6..10\n"

ruby| end

6..10
```

A continuación otro ejemplo con cadenas

```
ruby> case 'abcdef'
ruby| when 'aaa', 'bbb'
ruby| print "aaa o bbb\n"
ruby| when /def/
ruby| print "incluye /def/\n"
ruby| end
```

L9 Instrucciones Repetitivas

While

while <condición>

…. <bloque instrucciones separadas por líneas>

end

Existen cuatro formas de interrumpir el progreso de un ciclo desde su interior. La primera **break**, sale completamente del ciclo. La segunda **next** salta al principio de la siguiente iteración del bucle. La tercera **redo** reinicia la iteración en curso. La cuarta forma de salir del interior de un ciclo es **return**. La evaluación de return provoca la salida no sólo del bucle sino también del método o función conteniendo el bucle.

For

for <elem> **in** <coleccion>
…. <bloque instrucciones separadas por líneas>

end

A continuación se muestra un ejemplo

```
ruby> for num in (4..6)
ruby| print num, "\n"
ruby| end
4
5
6
```

En colección se puede utilizar cualquier estructura conteniendo valores. A continuación se muestra otro ejemplo.

```
ruby> for elm in [100,-9.6,"pickle"]

ruby| print "#{elm}\t(#{elm.type})\n"

ruby| end
```

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

100 (Fixnum)

-9.6 (Float)

pickle (String)

Iteradores

En Ruby existe el concepto de iteradores los cuales permiten iterar sobre los objetos empleando métodos específicos sobre estos.

A continuación se muestran algunos ejemplos, conteniendo iteradores sobre el tipo cadena.

```
ruby> "abc".each_byte{|c| printf"{%c}", c}; print "\n"
{a}{b}{c}

ruby> "a\nb\nc\n".each_line{|l| print l}
a
b
c
```

L10 Programación Orientada a Objetos

Clases

```
class <nombreclase>
    def <nombremetodo1>
        ... <bloque de instrucciones>
    end
    ...
    def <nombremetodon>
        ... <bloque de instrucciones>
    end
end
```

En Ruby se definen los métodos a través de funciones provistas en Ruby dentro del constructor **class** del lenguaje. Los campos de datos no se definen, una vez utilizados pasan a ser definidos.

A continuación se muestra un ejemplo.

```
ruby> class curso

ruby| def ca073

ruby| print "Curso Ruby On Rails \n"

ruby| end

ruby| end
```

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

Para instanciar un objeto de una clase se emplea el método **new**, a continuación se muestra un ejemplo:

```
ruby> cruby = curso.new

#<curso:0x401c444c>
```

Una vez creada la instancia de una clase, los métodos pueden ser invocados.

```
ruby> cruby.ca073
Curso Ruby On Rails
```

Herencia

En Ruby se implementa el concepto de herencia empleando la sintaxis **<**

```
ruby> class Herviboro
ruby| def alimenta
ruby| print "como vegetales y legumbres \n"
ruby| end
ruby| end
nil
ruby> class Conejo<Herviboro
ruby| def salta
ruby| print "desplaza con brincos \n"
ruby| end
ruby| end
nil

ruby> buggsbonny = Conejo.new
#<Conejo:0x401c41b8>
ruby> buggsbonny.alimenta
como vegetales y legumbres
nil
ruby> buggsbonny.salta
desplaza con brincos
nil
```

Purismo de Ruby en orientación por objetos:

Todos los campos de datos son privados, un objeto no puede acceder directamente a los datos sino es a través de los métodos. Un método definido en el contexto de una clase (pero no como método de la clase) no puede invocarse como un objeto. Por lo tanto los métodos privados se colocan como métodos fuera de la clase en el archivo donde se define la clase.

L11 Manejo de archivos

Ruby ofrece un conjunto de métodos de entrada y salida definidos en el módulo Kernel, todos estos métodos heredan de la clase IO.

La clase IO ofrece los métodos básicos para el manejo de archivos.

Abriendo y cerrando archivos:

Se puede crear un archivo para lectura o escritura o ambos empleando el método **new** y para cerrar el método **close**.

```
fp = File.new("nombre", "modo")

  # ... procesamiento
```

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

fp.close

También puede emplearse el método open para abrir un archivo

Los modos de apertura disponible son:

- "r" : solo lectura
- "r+": lectura y escritura
- "w": escritura, sino existe el archivo lo crea
- "w+": lectura y escritura, sino existe el archivo lo crea
- "a": solo escritura, si el archivo existe escribe al final del mismo
- "a+": lectura y escritura, si el archivo existe escribe al final del mismo

Procesando archivos:

El método **gets** lee una línea de archivo y desplaza el apuntador a la siguiente linea.

El método **sysread** con el parámetro "nc", permite leer el número de caracteres "nc" de un archivo y desplaza el apuntador en el número de caracteres leídos.

El método **syswrite** con el parámetro "datos", permite escribir los datos contenidos en "datos" en un y desplaza el apuntador.

El método **readlines** (de la clase IO) lee todo el contenido de un archivo y deja cada línea del mismo en una posición de un arreglo

arr = IO.readlines("archivo")

El método **rename**("viejo","nuevo") renombra un archive

El método **delete**("archivo") elimina un archivo

El método **chmod**("mascara") permite cambia la perisología de un archivo

L12 Módulos

Los módulos en Ruby son similares a las clases, excepto en:

• Un módulo no puede tener instancias

• Un módulo no puede tener subclases

• La sintaxis de construcción de un módulo se define con **module** ... **end**

Los módulos pueden ser utilizados para realizar programación estructurada en Ruby si no se desea utilizar programación orientada por objetos.

Existen dos usos típicos de los módulos. Uno es agrupar métodos y constantes relacionadas en un repositorio central.

El módulo Math de Ruby es un ejemplo de este caso:

ruby> Math.sqrt(2)

1.414213562

ruby> Math::PI

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

3.141592654

El operador **::** indica al intérprete de Ruby qué módulo debe consultar para obtener el valor de la constante (es concebible, que algún otro módulo a parte de Math interprete PI de otra forma). Si se requieres referenciar a los métodos o constantes de un módulo, directamente, sin utilizar ::, se incluye ese módulo con la instrucción **include**:

```
ruby> include Math

Object

ruby> sqrt(2)

1.414213562

ruby> PI

3.141592654
```

El otro uso de los módulos se denomina **mixin**.

L13 Excepciones

Rescue

En Ruby, como en muchos lenguajes modernos, se pueden gestionar las excepciones para bloques de código, permitiendo capturar los errores. El bloque de código marcado con **begin** se ejecutará hasta que se produzca una excepción, provocando que el control se transfiera al bloque de código de gestión de errores marcado con **rescue**. Si no se producen excepciones, el código de **rescue** no se utiliza. A continuación se muestra un ejemplo: el siguiente método devuelve la primera línea de un archivo de texto o nil si se produce una excepción.

```
def first_line( filename )
   begin
      file = open(filename)
      info = file.gets
      file.close
      info # Lo último que se evalúa es el valor devuelto
      rescue
          nil # No se pudo leer el archivo, no se devuelve nada
   end
end
```

Raise

Para disparar una excepción se puede utilizar la instrucción **raise**. Tiene un parámetro especificando la cadena describiendo la excepción. El parámetro es opcional pero no se debería omitir. Se puede acceder a él posteriormente a través de la variable global especial $!.

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

```
ruby> begin

ruby| raise "error"

ruby| rescue

ruby| print "Ha ocurrido un error: ", $!, "\n"

ruby| end

Ha ocurrido un error: error

nil
```

Ensure

Las instrucciones colocadas en el bloque ensure siempre se ejecutan se produzca o no una excepción. Esto puede ser útil para ejecutar labores de limpieza, cerrar un archivo abierto, liberar los datos de los buffers, etc.

```
begin

file = open("archivo","w")

# ... se escribe en el archivo ...

file.close

end
```

En el ejemplo anterior, si ocurre una excepción durante la parte del código donde se escribe en el archivo, éste quedará abierto.

```
begin

file = open("archivo","w")

# ... se escribe en el archivo...

rescue

# ... se gestionan las excepciones ...

ensure

file.close # ... Y esto se ejecuta siempre.

end
```

Se puede utilizar **ensure** sin **rescue** y viceversa, pero si se utilizan en el mismo bloque **begin** ... **end, rescue** debe preceder a **ensure**.

L14 Desarrollo de Aplicaciones WEB

Desarrollando scripts CGI

A continuación se muestra un script CGI en Ruby:

```
puts "HTTP/1.0 200 OK"
```

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

```
puts "Content-type: text/html\n\n"

puts "<html><body>Esto es una prueba</body></html>"
```

Este script puede almacenarse en un archivO prueba.cgi y puede procesarse en un servidor web procesando CGI. Por ejemplo si se alojara en el raíz del instituto gala

http://www.institutogala.com/prueba.cgi produciría una página HTMl mostrando Esto es una prueba.

Cuando la página test.cgi es requerida por el navegador, el servidor web busca por prueba.cgi en el servidor web; y lo ejecuta utilizando el interpretador de Ruby. El script de Ruby devuelve un encabezado http básico y luego devuelve el documento HTML.

Librería cgi.rb

Ruby ofrece una librería llamada cgi la cual permite el desarrollo de mayor funcionalidad CGI en las aplicaciones Web.

A continuación se muestra un ejemplo de su uso, reproduciendo el mismo ejemplo:

```
require 'cgi'

cgi = CGI.new

puts cgi.header

puts "<html><body>Esto es una prueba</body></html>"
```

Procesamiento de formularios:

Empleando la clase CGI se pueden tomar los parametros de un formulario HTML de dos maneras. Supongamos se tenga el URL /cgi-bin/prueba.cgi?Nombre=Rosa&Apellido=Perez.

Los parámetros Nombre y Apellido pueden obtenerse directamente con el siguiente script:

```
require 'cgi'

cgi = CGI.new

cgi['Nombre'] # => ["Rosa"]

cgi['LastName'] # => ["Perez"]
```

En el siguiente ejemplo se muestra otra forma de tomar los parámetros:

```
require 'cgi'

cgi = CGI.new

h = cgi.params # => {"Nombre"=>["Rosa"],"Apellido"=>["Perez"]}

h['Nombre'] # => ["Rosa"]

h['Apellido'] # => ["Perez"]
```

A continuación se muestra la manera de obtener todos los nombres de las variables (“keys”) almacenadas en HTML:

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

```
require 'cgi'

cgi = CGI.new

cgi.keys       # => ["Nombre", "Apellido"]
```

Si un formulario contiene múltiples campos con el mismo nombre, los valores correspondientes serán enviados al script en un arreglo.

Como ejemplo asuma que el formulario tiene tres campos llamado “Nombre” y los tres valores colocados son "Rosa", "Sara" y "Alicia":

```
require 'cgi'

cgi = CGI.new

cgi['Nombre']       # => "Rosa"

cgi.params['Nombre'] # => ["Rosa", "Sara", "Alicia"]

cgi.keys         # => ["Nombre"]

cgi.params       # => {"Nombre"=>["Rosa", "Sara", "Alicia"]}
```

Nota: Ruby maneja los métodos GET y POST automáticamente. No existe un tratamiento por separado para estos métodos.

Creando Formularios y HTML:

CGI ofrece varios métodos para crear HTML. Existe un método por cada tag. Para poder acceder a estos métodos, se deba crear un objeto CGI; invocando CGI.new.

A continuación se muestra un ejemplo

```
require "cgi"

cgi = CGI.new("html4")

cgi.out{

  cgi.html{

    cgi.head{ "\n"+cgi.title{"Esto es una prueba"} } +

    cgi.body{ "\n"+

      cgi.form{"\n"+

        cgi.hr +

        cgi.h1 { "Formulario: " } + "\n"+

        cgi.textarea("texto") +"\n"+

        cgi.br +

        cgi.submit
```

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

```
        }
      }
    }
}
```

NOTA: El método form de la clase CGI acepta como parámetro un método HTTP (GET, POST) para utilizar en el envío del formulario. El defecto, utilizado en este ejemplo es POST.

Este ejemplo produce el siguiente resultado:

```
Content-Type: text/html

Content-Length: 302

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Final//EN">

<HTML>

<HEAD>

<TITLE>Esto es una prueba </TITLE>

</HEAD>

<BODY>

<FORM METHOD="post" ENCTYPE="application/x-www-form-urlencoded">

<HR>

<H1>Formulario: </H1>

<TEXTAREA COLS="70" NAME="texto" ROWS="10"></TEXTAREA>

<BR>

<INPUT TYPE="submit">

</FORM>

</BODY>

</HTML>
```

L15 Desarrollo de aplicaciones en múltiples capas

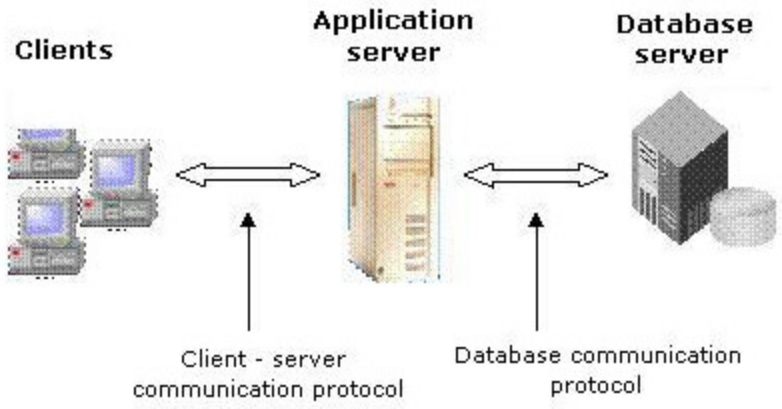
La idea principal de un sistema distribuido es la división lógica de la aplicación en varias capas, de forma de repartir las responsabilidades de realizar tareas específicas en cada una de ellas. En esta lección, las aplicaciones distribuidas se basan en una arquitectura de 3 capas, es decir, cada una de las capas se va a especializar en realizar determinadas tareas.

En la primera capa se encuentran los componentes de la aplicación implementando la interfaz de la misma con el cliente (Capa de Presentación), en la segunda se hayan los componentes ocupándose de ejecutar la lógica del negocio de la aplicación, todo lo que es comportamiento del sistema (Servidor de Aplicaciones), y en la tercera

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

capa están los componentes encargados de realizar toda la manipulación y persistencia de los datos (Servidor de Base de Datos).

A diferencia de las aplicaciones Cliente/Servidor tradicionales (2 capas), donde la ejecución de todo el código de la aplicación (lógica del negocio) se realiza en el cliente, en una aplicación 3 capas, se distribuye el código ejecutando parte en el cliente y parte en el servidor de aplicaciones. De esta forma se logra ganar en escalabilidad, seguridad y rendimiento.



Cuando el cliente requiere algo (por ejemplo datos), se comunica con el servidor de aplicaciones solicitándole ejecutar determinado proceso remoto, con determinados parámetros. El servidor de aplicaciones dispara dicho proceso, y luego se comunica con el cliente para retornarle el resultado correspondiente.

¿Por qué implementar una aplicación en 3 capas?

En una arquitectura tradicional en 2 capas (cliente/servidor), toda la lógica de la aplicación se ejecuta en el cliente y el acceso a la base de datos se realiza desde el cliente. En este caso, se debe tener clientes con alto potencial, capaces de ejecutar la aplicación con un alto rendimiento; además, cada uno de estos clientes debería estar configurado para acceder a la base de datos con el driver correspondiente al manejador de base de datos utilizado.

L16 Framewoks de desarrollo

El concepto framework se emplea en muchos ambientes del desarrollo de sistemas software, no solo en el ámbito de aplicaciones Web. Podemos encontrar frameworks para el desarrollo de aplicaciones médicas, de visión por computador, para el desarrollo de juegos, y en general para cualquier ambiente.

En general, el término framework, se refiriere a una estructura de software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un framework se puede considerar como una aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta.

Los objetivos principales que persigue un framework son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones.

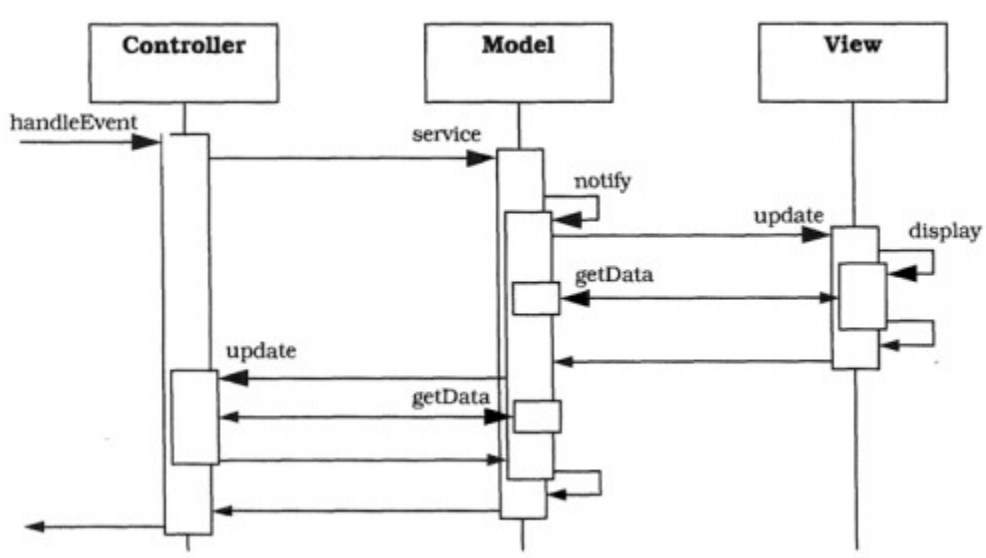
Un framework Web, por tanto, podemos definirlo como un conjunto de componentes formando un diseño reutilizable para facilitar y agilizar el desarrollo de sistemas Web.

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

Para comprender como trabajan los frameworks Web existentes es imprescindible conocer el patrón **MVC**; **M**odelo **V**ista **C**ontrolador.

El patrón Modelo-Vista-Controlador es una guía para el diseño de arquitecturas de aplicaciones ofreciendo una fuerte interactividad con usuarios. Este patrón organiza la aplicación en tres modelos separados, el primero es un modelo representando los datos de la aplicación y sus reglas de negocio, el segundo es un conjunto de vistas representando los formularios de entrada y salida de información, el tercero es un conjunto de controladores procesando las peticiones de los usuarios y controlando el flujo de ejecución del sistema.

En el siguiente diagrama de secuencia de UML se especifica el funcionamiento del modelo MVC.



Las siguientes funciones se encuentran por lo general disponibles en un framework para desarrollo web:

- Abstracción de URLs y sesiones: No es necesario manipular directamente las URLs ni las sesiones, el framework se encarga de hacerlo.
- Acceso a datos: Incluyen las herramientas e interfaces necesarias para integrarse con herramientas de acceso a datos.
- Controladores: La mayoría de frameworks implementa una serie de controladores para gestionar eventos, como la introducción de datos mediante un formulario o el acceso a una página. Estos controladores suelen ser fácilmente adaptables a las necesidades de un proyecto concreto.
- Autenticación y control de acceso: Incluyen mecanismos para la identificación de usuarios mediante login y password y permiten restringir el acceso a determinadas páginas a determinados usuarios.
- Internacionalización.
- Separación entre diseño y contenido.

L17 ¿Qué es Rails?

Rails en un framework para el desarrollo de aplicaciones Web escrito en el lenguaje de programación Ruby. El objetivo del diseño de Ruby es facilitar el desarrollo de aplicaciones web ofreciendo un conjunto de herramientas necesarias para iniciar el desarrollo de cualquier tipo de aplicación. Permite desarrollar la aplicación en un tiempo corto y con un menor número de líneas de código.

Rails se basa en la asunción de que existe el mejor camino para realizar las cosas y se encuentra diseñado para soportar la utilización de ese camino y para evitar la toma de caminos incorrectos. Si se realizan las cosas en el

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

modo “Rails” se obtienen incrementos en la productividad. “Rails” debe verse como un paradigma y un patrón de programación.

La filosofia de Rails incluye los siguientes principios básicos:

- **DRY** – “Don’t Repeat Yourself” – escribir el mismo código una y otra vez es una mala idea.
- Convención sobre Configuración– Rails realiza asunciones acerca de lo que se quiere realizar y como se va a realizar en lugar de que el programador escriba todos los detalles sobre lo que quiere realizar empleando un sin fin de archivos de configuración.
- **REST** es el mejor patron para el desarrollo de aplicaciones web – organizar la aplicación alrededor de recursos y standard HTTP es el mejor camino a seguir.

L18 Instalación de Rails: Windows, Linux

En la mayoría de los casos, el modo más expedito de instalar Rails es utilizar RubyGems:

Se puede ejecutar desde el usuario root:

```
# gem install rails
```

En el caso de Windows, se recomienda instalar Ruby y Rails con el instalador de Rails. Este se encuentra en el siguiente URL http://railsinstaller.org/

Para verificar que la instalación fue realizada satisfactoriamente se puede ejecutar en la línea de comandos:

```
$ rails --version
```

Si se visualiza en la pantalla algo como “Rails x.y.z” indica la instalación se ha realizado correctamente

L19 Framework Ruby On Rails: Modelo, Vista, Controlador

La base de Rails es la arquitectura Modelo, Vista, Controlador conocida como MVC. Entre los beneficios de MVC se tiene:

- Separación de la lógica del negocio de la interfaz del usuario
- Evita repetir código, concepto DRY
- Clarifica la ubicación del código fuente , facilitando el mantenimiento

Modelos

Un modelo representa la información (datos) de la aplicación y las reglas para manipular los datos. En el caso de Rails, los modelos son principalmente utilizados para administrar las reglas de interacción con una base de datos. En la mayoría de los casos, cada tabla de la base de datos se corresponderá con un modelo en la aplicación. Casi la totalidad del código de la lógica del negocio de la aplicación se concentra en los modelos.

Vistas

Las vistas representan las interfaces de las aplicaciones. En Rails, las vistas con archivos HTML con código Ruby embebido ejecutando las tareas únicamente relacionadas con la presentación de los datos. Las vistas se encargan del trabajo de suministrar los datos al navegador web u a otra herramienta utilizada para realizar los requerimientos de la aplicación.

Controladores

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

Los controladores proveen el "pegamento" entre los modelos y las vistas. En Rails, los controladores son los responsables de procesar los requerimientos recibidos desde el navegador web, interrogar a los modelos para obtener los datos y transmitir los datos a las vistas para la presentación.

Componentes de Rails

Rails ofrece un conjunto de componentes con su instalación. Cada uno de estos componentes, será brevemente explicado a continuación.

- Action Pack
  - Action Controller
  - Action Dispatch
  - Action View
- Action Mailer
- Active Model
- Active Record
- Active Resource
- Active Support
- Railties

Action Pack

Action Pack empaqueta la parte VC del MVC con los componentes Action Controller, Action View y Action Dispatch.

Action Controller

Action Controller es el componente que administra los controladores en una aplicación Rails. Procesa los requerimientos recibidos por una aplicación rails, extrae los parámetros e invoca la acción requerida. Los servicios ofrecidos por el Action Controller incluyen: el manejo de las sesiones, el despliegue de las plantillas y el control del redirecionamiento.

Action View

Action View administra las vistas de una aplicación Rails. Puede crear tanto archivos HTML como XML de salida. Action View administra el despliegue de las plantillas incluyendo las plantillas anidadas y las plantillas parciales; también ofrece soporte AJAX.

Action Dispatch

Action Dispatch maneja el enrutamiento de los requerimientos web y los despacha según la solicitud.

Action Mailer

Action Mailer es un componente para construir servicios de correo electrónico.

Active Model

Active Model provee una interface entre los servicios de Action Pack y los mapeos de bases de datos Object Relationship Mapping (ORM) tales como Active Record. Active Model permite a Rails utilizar otros ORM en lugar de Active Record si la aplicación lo requiere.

Active Record

Active Record es la base para los modelos en una aplicación Rails. Provee independencia de la Base de Datos, la funcionalidad CRUD (Create, Read, Update, Delete) , capacidades de búsqueda avanzadas y la habilidad de relacionar modelos..

Active Resource

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

Active Resource es un componente utilizado para gerenciar la conexión entre los objetos del negocio y los servicios web RESTful. Implementa un mecanismo para mapear los recursos web con objetos locales con la semántica CRUD.

Active Support

Active Support es una colección extensa de clases utilitarias y librerías Ruby standard utilizadas en Rails.

Railties

Railties es el código base generado por Rails al construir nuevas aplicaciones.

L20 Estructura directorios Rails

Al crear una nueva aplicación, Rails crea una estructura de directorios.

Por ejemplo si se crea una aplicación llamada gala con el siguiente comando

$ rails new gala

Se crea un directorio gala y debajo de este, la siguiente estructura de directorios y archivos:

Archivo / Directorio	Propósito
app/	Contiene los controladores, modelos, vistas y activos de la aplicación.
config/	Configuración en tiempo real de la aplicación: reglas, rutas, bases de datos y otros aspectos.
config.ru	Archivo de configuración de los racks en el caso de servidores en rack.
db/	Contiene el esquema actual de la base de datos y las migraciones de las bases de datos.
doc/	Documentación de la aplicación.
Gemfile Gemfile.lock	Estos archivos especifican las dependencias gem para la aplicación Rails.
lib/	Módulos extendidos para la aplicación.
log/	Archivos de log de la aplicación.
public/	Directorio de publicación de la aplicación. Contiene archivos estáticos y activos compilados.

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

Rakefile	Contiene o realiza la carga de las tareas a ser ejecutadas desde la línea de comandos. Las definiciones de tareas son indicadas a través de los componentes de Rails. En lugar de modificar el archivo Rakefile, se debe agregar las tareas creando archivos en el directorio lib/tasks de la aplicación.
README.rdoc	Manual de instrucciones de uso rápido de la aplicación.
script/	Contiene los scritps rails para arrancar la aplicación y puede alojar otros scripts de implantación .
test/	Artifactos para realizar las pruebas de las aplicaciones.
tmp/	Archivos temporales.
vendor/	Un lugar para colocar el código de terceras partes . En una aplicación típica de rails contiene, ruby gems, el código fuente de rails y otros plugins.

L21 Desarrollo aplicaciones en Rails

Rails ofrece un conjunto de utilitario para el desarrollo de aplicaciones.

Al crear una nueva aplicación, Rails crea una estructura de directorios.

Por ejemplo si se crea una aplicación llamada gala con el siguiente comando

$ rails new gala

Se crea un directorio gala y debajo de este, la estructura de directorios indicada en la lección anterior.

Arrancando el servidor web

Una vez creada la aplicación la misma puede ser probada en el ambiente de desarrollo levantando el servidor de web de Rails. Esto se realiza ejecutando el siguiente comando:

$ rails server

Este comando iniciara una instancia del servidor web **WEBrick** por defecto (Rails también puede utilizar otros servidores web. Para visualizar la aplicación en ejecución se puede abrir un navegador web y colocar el URL **http://localhost:3000** .

Creando un controlador y una vista básica en Rails

Rails ofrece un comando simple para realizar esta tarea, se puede ejecutar desde la línea de comando:

$ rails generate controller home index

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

Rails creara varios archivos de manera automática, incluyendo app/views/home/index.html.erb. Este archivo contiene la plantilla a ser utilizada para desplegar el índice en la página de inicio del controlador. Por ejemplo este archivo puede ser editado y escribir:

```
<h1>Hola, Mundo !</h1>
```

Para que Rails tome esta página como la página defecto de la aplicación debe eliminarse la página índice por defecto ubicada en el directorio public

```
$ rm public/index.html
```

Luego debe indicarse en Rails donde se encuentra ubicada la página actual inicio. Se debe abrir el archivo config/routes.rb , luego se debe editar la línea iniciando con root :to y eliminar el carácter de inicio de comentario #:

```
  #...

  # You can have the root of your site routed with "root"

  # just remember to delete public/index.html.

  root :to => "home#index"
```

La linea root :to => "home#index" le indica a Rails que mapee la acción de raiz con el inicio del controlador.

Si ahora se coloca en el navegador http://localhost:3000, se visualizara “Hola, Mundo!”.

L22 Configuración de Bases de Datos para aplicaciones Rails

La base de datos a utilizar por la aplicación es especificada en el archivo de configuración, **config/database.yml.** Si este archivo es abierto en una aplicación creada con Rails se presentara una configuración por defecto para SQLite. El archivo contiene la configuración para los tres ambientes que Rails puede ejecutar por defecto:

- El ambiente de desarrollo con el cual se puede interactuar de manera manual con la aplicación
- El ambiente de pruebas, el cual es utilizado para ejecutar pruebas automáticas
- El ambiente de producción.

NOTA: No es necesario actualizar las configuraciones de las Bases de Datos de manera manual. En las opciones del generador de aplicaciones existe una opción denominada—database. Esta opción permite escoger un adaptador de una lista de los manejadores de bases de datos relacionales más utilizados. Por ejemplo si se ejecuta rails new gala —database=mysql; cuando se confirme la sobreescritura del archivo config/database.yml, la aplicación se configurara para MySQL en lugar de SQLite. A continuación se muestran ejemplos para SQLLite3, mysql y Postgresql.

Base de Datos SQLite3

```
development:

  adapter: sqlite3

  database: db/development.sqlite3

  pool: 5

  timeout: 5000
```

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

Base de Datos MySQL

```
development:
  adapter: mysql2
  encoding: utf8
  database: gala_development
  pool: 5
  username: root
  password:
  socket: /tmp/mysql.sock
```

Base de Datos PostgreSQL

```
development:
  adapter: postgresql
  encoding: unicode
  database: gala_development
  pool: 5
  username: gala
  password:
```

Creando la Base de Datos

Una vez configurada la Base de Datos, Rails puede crear una base de datos vacía empleando el comando **rake**.

```
$ rake db:create
```

Esto creara las bases de datos para los ambientes de desarrollo y de pruebas en el directorio db/ folder.

Rake es un comando de propósito general empleado por Rails para ejecutar varias acciones. Se puede visualizar la lista de todas las posibilidades de rake ejecutando rake -T.

L23 Capa de Modelo en Rails

Para crear un modelo en Rails se emplea el siguiente comando

```
rails generate model Person name:string
```

Este ejemplo crea un modelo Person con el campo name del tipo string, los campos se separan por espacios en blancos.

A continuación se muestra el resultado de la ejecución del comando

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

```
C:\Sites\railsinstaller_demo>rails generate model Person name:string
        SECURITY WARNING: No secret option provided to Rack::Session::Cookie.
        This poses a security threat. It is strongly recommended that you
        provide a secret to prevent exploits that may be possible from crafted
        cookies. This will not be supported in future versions of Rack, and
        future versions will even invalidate your existing user cookies.

        Called from: C:/RailsInstaller/Ruby1.9.3/lib/ruby/gems/1.9.1/gems/action
pack-3.2.1/lib/action_dispatch/middleware/session/abstract_store.rb:28:in `initi
alize'.
      invoke  active_record
      create    db/migrate/20130215230625_create_people.rb
      create    app/models/person.rb
      invoke    test_unit
      create      test/unit/person_test.rb
      create      test/fixtures/people.yml
```

Los modelos en rails se basan en el Modulo ActiveRecord, el cual tiene los siguientes módulos:

- ActiveRecord::Aggregations
- ActiveRecord::Associations
- ActiveRecord::AttributeAssignment
- ActiveRecord::AttributeMethods
- ActiveRecord::AutosaveAssociation
- ActiveRecord::Batches
- ActiveRecord::Calculations
- ActiveRecord::Callbacks
- ActiveRecord::Coders
- ActiveRecord::ConnectionAdapters
- ActiveRecord::CounterCache
- ActiveRecord::Delegation
- ActiveRecord::DynamicMatchers
- ActiveRecord::Explain
- ActiveRecord::FinderMethods
- ActiveRecord::IdentityMap
- ActiveRecord::Inheritance
- ActiveRecord::Integration
- ActiveRecord::Locking
- ActiveRecord::ModelSchema
- ActiveRecord::NestedAttributes
- ActiveRecord::Persistence
- ActiveRecord::QueryMethods
- ActiveRecord::Querying
- ActiveRecord::Railties
- ActiveRecord::ReadonlyAttributes
- ActiveRecord::Reflection
- ActiveRecord::Sanitization
- ActiveRecord::Scoping
- ActiveRecord::Serialization
- ActiveRecord::SpawnMethods
- ActiveRecord::Store
- ActiveRecord::TestFixtures
- ActiveRecord::Timestamp
- ActiveRecord::Transactions
- ActiveRecord::Translation
- ActiveRecord::Validations

Active Record

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

El componente Active Record es la capa Object/Relational Mapping (ORM) de Rails. Este componente sigue estrictamente el modelo ORM model:

- Las Tablas se mapean en clases
- Los Registros de mapean en objetos
- Las Columnas se mapean en los campos de los objetos

Los Active Records de Rails proven una interface y un binding entre las tablas de una base de datos relacional y el código Ruby manipulando los registros de la Base de Datos. Los nombres de los métodos Ruby son generados automáticamente de los nombres de los campos de las tablas de las bases de datos.

Cada objeto Active Record tiene los metodos CRUD (Create, Read, Update, and Delete) para acceder a la Base de Datos.

Traduciendo un modelo en SQL

Las siguientes reglas deben seguirse:

- Cada entidad (ejemplo libro) se asocia con una tabla de la base de datos pero en plural (libros).
- Cada entidad con una tabla debe tener in campo único entero denominado id
- Si existe una relación entre las entidades x y y; y la entidad y hace una referencia a la entidad x, la tabla y debe tener una clave foránea y la columna debe llamarse x_id
- Cada columna de la tabla representa un campo de la entidad

Creando los archivos de Active Record

Se debe generar un modelo por cada tabla

$ rails generate model Book

$ reils generate model Subject

Por convención se coloca la primera letra de los modelos en mayúscula y las tablas deben estar en minúscula y en plural.

Este proceso crea varios archivos, pero los archivos de los modelos se crean en el directorio app/models book.rb y subject.rb.

En el archivo book.rb

```
class Book < ActiveRecord::Base

end
```

En el archivo subject.rb

```
class Subject < ActiveRecord::Base

end
```

Creando asociaciones entre los modelos:

Al igual que en el modelo entidad-relacion los Active Record soportan tres tipos de asociaciones: uno a uno. Uno a muchos, muchos a muchos.

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

Estas asociaciones se especifican en los modelos agregando declaraciones: has_one, has_many, belongs_to, and has_and_belongs_to_many.

Como un book se relaciona con un subject y un subject puede estar en varios books (fijarse en los singular y plural de la declaración), la declaracion se especifica de la siguiente manera:

```
class Book < ActiveRecord::Base

  belongs_to :subject

end


class Subject < ActiveRecord::Base

  has_many :books

end
```

ActiveRecord::Validations

A continuación se muestra una guía del uso de **ActiveRecord::Validations**

Las Validaciones son utilizadas para asegurar que únicamente datos validos son almacenados en las bases de datos.

Existen varios modos de validar los datos antes de ser almacenados en la base de datos: restricciones nativas de las bases de datos, validaciones en el cliente, validaciones en el controlador y validaciones en el modelo.

Existen dos clases de objetos Active Record: aquellos representando un registro de base de datos y aquellos que no. Cuando se crea un objeto nuevo, el objeto no pertenece a la base de datos todavía. Una vez guardado el objeto este será almacenado en la tabla de base de datos correspondiente. Active Record utiliza el metodo new_record? Para determinar si el objeto se encuentra en la Base de Datos. Considere el siguiente ejemplo de modelo:

```
class Person < ActiveRecord::Base

end


>> p = Person.new(:name => "John Doe")

=> #<Person id: nil, name: "John Doe", created_at: nil, :updated_at: nil>

>> p.new_record?

=> true

>> p.save

=> true

>> p.new_record?

=> false
```

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

Crear y guardar un nuevo objeto, enviara una operación SQL INSERT a la base de datos. Actualizando un objeto existente enviara una operación SQL UPDATE. Las validaciones son ejecutadas antes de que los comandos sean enviados a la Base de Datos. Si cualquiera de las validaciones falla, el objeto será marcado como invalido y Active Record no podrá ejecutar la operación INSERT o UPDATE.

Los siguientes metidos disparan validaciones y los objetos serán registrados si el objeto es válido:

create

create!

save

save!

update

update_attributes

update_attributes!

Los métodos conteniendo el **sufijo !** disparan una excepción si el registro es invalido.

Los siguientes métodos obvian las validaciones y registraran el objeto independientemente de ser válido o no.

decrement!

decrement_counter

increment!

increment_counter

toggle!

touch

update_all

update_attribute

update_column

update_counters

El método **save** también puede obviar la validación si se indica

```
save(:validate => false)
```

Para verificar cuando un objeto es válido o no Rails utiliza el método **valid?.**

```
class Person < ActiveRecord::Base
  validates :name, :presence => true
end
```

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

```
Person.create(:name => "John Doe").valid? # => true

Person.create(:name => nil).valid? # => false
```

Una vez realizada la validación los errores pueden ser visualizados empleando el método errors.

```
class Person < ActiveRecord::Base
  validates :name, :presence => true
end
```

El método new no invoca la validación, por lo tanto aunque existan errores se mostrara en vacío.

```
>> p = Person.new
=> #<Person id: nil, name: nil>
>> p.errors
=> {}

>> p.valid?
=> false
>> p.errors
=> {:name=>["can't be blank"]}

>> p = Person.create
=> #<Person id: nil, name: nil>
>> p.errors
=> {:name=>["can't be blank"]}

>> p.save
=> false

>> p.save!
=> ActiveRecord::RecordInvalid: Validation failed: Name can't be blank

>> Person.create!
=> ActiveRecord::RecordInvalid: Validation failed: Name can't be blank
```

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

Retomando en ejemplo de los libros

```
class Book < ActiveRecord::Base
  belongs_to :subject
  validates_presence_of :title
  validates_numericality_of :price, :message=>"Error Message"
end
```

validates_presence_of – evita campos "NOT NULL"

validates_numericality_of – evita introducir datos no numericos

L24 Capa de Vista en Rails

Creando las respuestas

Desde el punto de vista del controlador existen tres maneras de crear una respuesta HTTP:

- Ejecutar **render** para crear una respuesta complete a ser enviada al navegador
- Ejecutar **redirect_to** para enviar un estatus de redirección de HTTP al navegador
- Ejecutar **head** para enviar cabeceras HTTP al navegador

Despliegue por Defecto: Convención sobre Configuración en acción

Por defecto, los controladores en Rails automáticamente despliegan vistas con los nombres que corresponden a rutas validas, Por ejemplo si el código de la clase CursosController es el siguiente:

```
class CursosController < ApplicationController
end
```

Y se encuentra la siguiente línea en el archivo de rutas:

```
resources : cursos
```

Y existe el siguiente archivo de vista app/views/cursos/index.html.erb:

```
<h1>Bienvenido al Instituto Gala !</h1>
```

Rails automáticamente desplegara app/views/cursos/index.html.erb cuando se navegue a /cursos y se desplegara "Bienvenido al Instituto Gala!" en la pantalla.

En el siguiente ejemplo se mostrara como crear un modelo de cursos y agregar la acción de índice al controlador:

```
class CursosController < ApplicationController
  def index
```

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

```
    @cursos = Cursos.all

  end

end
```

La regla de rails es que si no se indica explícitamente que se va a desplegar al final de una acción de controlador, Rails automáticamente desplegara lo que se encuentre en el archivo nombre_accion.html.erb en el camino del controlador. En este caso, Rails desplegara el archivo app/views/cursos/index.html.erb file.

Si se requieren desplegar todas las propiedades de todos los cursos, se puede crear una plantilla erb como esta:

```
<h1>Listado de Cursos</h1>


<table>

  <tr>

    <th>Codigo</th>

    <th>Resumen</th>

    <th></th>

    <th></th>

    <th></th>

  </tr>


<% @cursos.each do |curso| %>

  <tr>

    <td><%= curso.codigo %></td>

    <td><%= curso.content %></td>

    <td><%= link_to 'Mostrar', curso %></td>

    <td><%= link_to 'Edit', edit_curso_path(curso) %></td>

    <td><%= link_to 'Remove', curso, :confirm => 'Are you sure?', :method => :delete %></td>

  </tr>

<% end %>

</table>


<br />

<%= link_to 'Nuevo curso', new_curso_path %>
```

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

El despliegue actual es realizado por subclases de la clase **ActionView::TemplateHandlers**. Es importante destacar que la extension de los archivos controla la escogencia del manejador de la plantilla. A partir de Rails 2, las extensions son **.erb** para ERB (HTML con Ryby embebido), and **.builder** para Builder (un generador de XML).

Modo 1 , ejecutando render

En muchos casos, el metodo **ActionController::Base#render** realiza la tarea pesada de desplegar el contenido a mostrar por el navegador. Existen muchas maneras de personalizar el comportamiento de render. Se puede desplegar la vista defecto de una plantilla Rails, o una plantilla específica, o un archivo, o código en línea, o nada. También se puede desplegar texto, JSON, o XML. También se puede especificar el tipo de contenido o el estatus HTTP de la respuesta.

Si se requieren mostrar los resultados exactos de una llamada a render sin necesidad de inspeccionar en el navegador se puede utilizar **render_to_string**.

La siguiente línea se utiliza para **desplegar nada**

render :nothing => true

Si se utiliza el utilitario **cURL**, se obtendrá lo siguiente:

```
$ curl -i localhost:3000/cursos

HTTP/1.1 200 OK

Connection: close

Date: Wed, 13 Feb 2013 09:25:18 GMT

Transfer-Encoding: chunked

Content-Type: */*; charset=utf-8

X-Runtime: 0.014297

Set-Cookie: _blog_session=...snip...; path=/; HttpOnly

Cache-Control: no-cache
```

El siguiente ejemplo muestra el despliegue de una **vista de una acción**

```
def update
  @curso = Curso.find(params[:id])
  if @curso.update_attributes(params[:curso])
    redirect_to(@curso)
  else
    render "edit"
```

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

```
  end
end
```

Si la llamada a update_attributes falla, invocar a la acción update en este controlador mostrara la plantilla edit.html.erb perteneciente al mismo controlador.

También puede ser utilizado un símbolo en lugar de una cadena para especificar la acción a mostrar:

```
def update
  @curso = Curso.find(params[:id])
  if @curso.update_attributes(params[:curso])
    redirect_to(@curso)
  else
    render :edit
  end
end
```

De una manera explícita, se puede utilizar la opción :action (esto no es necesario a partir de Rails 3.0):

```
def update
  @curso = Curso.find(params[:id])
  if @curso.update_attributes(params[:curso])
    redirect_to(@curso)
  else
    render :action => "edit"
  end
end
```

Using render with :action is a frequent source of confusion for Rails newcomers. The specified action is used to determine which view to render, but Rails does not run any of the code for that action in the controller. Any instance variables that you require in the view must be set up in the current action before calling render.

Existen otras opciones de despliegue con render in Rails:

- de una plantilla de acción de otro controlador
- Mostrando un archivo arbitrario
- A través de inline, código embebido
- Mostrando texto
- JSON
- XML
- Javascript

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

Modo 2 , ejecutando redirect_to

El método **redirect_to** le indica al navegador que procese un nuevo requerimiento especificando un URL.

redirect_to photos_url

Se puede utilizar redirect_to con argumentos, por ejemplo a continuación se muestra el caso de regresar a la página anterior

redirect_to :back

Modo 3 , ejecutando head

El método **head** puede ser utilizado para enviar respuestas con solo encabezados al navegador. El método head solo toma un parámetro, el cual puede ser interpretado como una tabla de hash con los nombres de los encabezados y los valores. Por ejemplo, se puede retornar un encabezado de error:

head :bad_request

Esto, producirá el siguiente encabezado:

HTTP/1.1 400 Bad Request

Connection: close

Date: Wed, 13 Feb 2013 12:15:53 GMT

Transfer-Encoding: chunked

Content-Type: text/html; charset=utf-8

X-Runtime: 0.013483

Set-Cookie: _blog_session=...snip...; path=/; HttpOnly

Cache-Control: no-cache

También se pueden emplear otros encabezados HTTP para manipular la información:

head :created, :location => photo_path(@photo)

Lo cual producira :

HTTP/1.1 201 Created

Connection: close

Date: Wed, 13 Feb 2013 12:16:44 GMT

Transfer-Encoding: chunked

Location: /photos/1

Content-Type: text/html; charset=utf-8

X-Runtime: 0.083496

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

Set-Cookie: _blog_session=...snip...; path=/; HttpOnly

Cache-Control: no-cache

L25 Capa de Controlador en Rails

El componente **Action Controller** es la C del MVC; en Controlador. Después que el enrutador determina cual es el controlador a utilizar para responder a un requerimiento, el Controlador es el responsable de atenderlo y de producir la salida apropiada. Afortunadamente, el Action Controller realiza la mayoría del trabajo pesado y emplea convenciones inteligentes realizando esta tarea lo más fácil posible.

En la mayoría de las aplicaciones, el controlador recibirá el requerimiento (esto es transparente para el desarrollador), tomara datos o enviara datos desde el modelo y utilizara una vista para crear la salida HTML.

Un controlador puede ser pensado como el intermediario entre los modelos y las vistas. Permite que los datos del modelo estén visibles para las vistas y así se puede desplegar los datos al usuario, y actualice los datos registrados por el usuario en el modelo.

Métodos y Acciones

Un controlador es una clase en Ruby heredando de **ApplicationController** y contiene métodos como cualquier otra clase. Cuando la aplicación recibe un requerimiento, el enrutador determina cual controlador y acción a ejecutar, entonces Rails crea una instancia del controlador y ejecuta el método con el mismo nombre de la acción.

```
class ClientsController < ApplicationController

  def new

  end

end
```

A manera de ejemplo, si el usurio indica /clients/new en la aplicación para crear un nuevo cliente, Rails creara una instancia de la clase ClientsController y ejecutara el método new. El método puede no tener código, en este caso Rails desplegara la página de vista new.html.erb. El método new podría crear un nuevo cliente y hacerlo disponible a la vista a traves de la instancia @client:

```
def new

  @client = Client.new

end
```

ApplicationController hereda de **ActionController::Base**, definiendo varios métodos útiles.

Solo los métodos públicos pueden ser invocados como acciones. Es una mejor práctica ocultar los métodos que no deberían ser acciones como los métodos auxiliares o los filtros.

Parámetros

Es necesario accede a los datos enviados por el usuario u otros parámetros en las acciones del controlador. Existen dos tipos de parámetros en las aplicaciones web. Los primeros , son los enviados por el URL conocidos como “query string”. Los segundos son los conocidos como los tipos POST en un formulario. Rails no realiza distinción entre estos tipos de parámetros y todos se encuentran disponible en la tabla de hashing **params** en el controlador:

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

```
class ClientsController < ActionController::Base

  #  ejemplo un query string /clients?status=activated

  def index

    if params[:status] == "activated"

      @clients = Client.activated

    else

      @clients = Client.unactivated

    end

  end


  # ejemplo de un formulario con "/clients"

  def create

    @client = Client.new(params[:client])

    if @client.save

      redirect_to @client

    else

      # Esta línea sobreescribe la acción de despliegue defecto

      render :action => "new"

    end

  end

end
```

L26 Layouts en Rails

Cuando Rails despliega una vista como una respuesta, lo realiza combinando la vista con el layout actual, empleando las reglas para encontrar el layout actual. Dentro de un layout, se tienen acceso a tres herramientas para combinar diferentes piezas de salida para construir la respuesta final:

- Tags Helpers (considerados como activos)
- Yield y content_for
- Partials

Tag Helpers

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

Tag helpers ofrecen métodos para generar HTML enlazando las vistas a feeds, Javascript, hokas de estilo, imagines, videos y audio. Existen seis tipos de tag helpers disponibles en Rails:

- auto_discovery_link_tag
- javascript_include_tag
- stylesheet_link_tag
- image_tag
- video_tag
- audio_tag

Se pueden emplear estos tags en layouts u otras vistas, aunque los tags auto_discovery_link_tag, javascript_include_tag, y stylesheet_link_tag, por lo general se utilizan en la sesión <head> de un layout.

Yield

Dentro del contexto de un layout, un yield identifica una sesión donde el contenido de la vista debería ser insertado, La manera más simple de utilizar es tener un solo yield dentro del cual la totalidad del contenido de la vista a ser desplegado es insertado.

```
<html>
  <head>
  </head>
  <body>
  <%= yield %>
  </body>
</html>
```

Se puede crear un layout con múltiples regiones de yield:

```
<html>
  <head>
  <%= yield :head %>
  </head>
  <body>
  <%= yield %>
  </body>
</html>
```

El cuerpo principal de la vista siempre será desplegado en el yield sin nombre. Para desplegar el contenido en un yield con nombre se debe utilizar el método content_for.

Metodo content_for

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

El metodo **content_for** permite insertar contenido en un bloque de un yield con nombre en un layout. Por ejemplo esta vista trabajara con el layout acabado de mostrar:

```
<% content_for :head do %>
  <title>A simple page</title>
<% end %>

<p>Hello, Rails!</p>
```

El resultado de desplegar esta página en el layout es este documento HTML:

```
<html>
  <head>
  <title>A simple page</title>
  </head>
  <body>
  <p>Hello, Rails!</p>
  </body>
</html>
```

El método **content_for** es muy útil cuando el layout contiene distintas regiones como barras laterales y pies de página y requieren bloques de contenidos insertados. Es también útil para insertar tas que cargan páginas específicas de JavaScript o archivos css en el encabezado o en otro layout genérico.

Utilizando partials

Las plantillas parciales – generalmente llamadas “partials” – es otro mecanismo para descomponer el proceso de despliegue en trozos más manejables. Con un partial, se puede mover el código para una parte de una respuesta a su propio archivo.

A continuación se muestra un ejemplo, donde ad_banner y footer son los partials.

```
<%= render "shared/ad_banner" %>

<h1>Products</h1>

<p>Here are a few of our fine products:</p>
...

<%= render "shared/footer" %>
```

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

L27 Desarrollo de Prototipos en Rails

Rails provee una serie de facilidades para desarrollar prototipos de una manera rápida. Viene con defecto con una base de datos sqllite, ofrece un servidor web por defecto y viene con varias plantillas para la creación de los modelos, de las vistas y de los controladores.

La separación en capas permite luego migrar a cualquier manejador de base de datos relacional de los más utilizados y realizar la implementación en un servidor web en producción.

Los modelos con las facilidades de CRUD permiten desarrollar prototipos funcionales sin ejecutar una sentencia SQL.

L28 Combinando Rails con AJAX

Rails 3.1 viene con jQuery como la librería JavaScript defecto. Gemfile contiene gem 'jquery-rails' poniendo a disposición los archivos jQuery para la aplicación automáticamente. Esto puede ser accedido como:

javascript_include_tag :defaults

AJAX Rails Helper: link_to_remote

El helper **link_to_remote** tiene una característica interesante desde el punto de vista de la documentación; las opciones de link_to_remote son compartidas por los otros helpers de AJAX helpers.

La firma de link_to_remote se indica a continuación:

```
def link_to_remote(name, options = {}, html_options = nil)
```

A continuacion se muestra un ejemplo de link_to_remote:

```
link_to_remote "Add to cart",
  :url => add_to_cart_url(product.id),
  :update => "cart"
```

El primer parámetro es una cadena conteniendo el texto del link que se muestra en la página.

El segundo parámetro es una tabla de hash e indica la especificación de AJAX.:

:url es el parámetro necesario para generar el enlace remoto. Este url apunta al manejador de acción de AJAX.

:update especifica el id DOM del elemento que se requiere actualizar

Formularios AJAX

Existen tres formas diferentes de agregar formularios AJAX a las vistas de Rails utilizando helpers. Estas tres son ligeramente diferentes, pero lograr el mismo objetivo no requiere la recarga de la página.

- **remote_form_for** (y su alias **form_remote_for**) : toma un recurso, modelo o arreglo de recursos como parámetro.

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

- **form_remote_tag** : toma los datos de formulario, los serializa y envía sus datos en el background
- **submit_to_remote** y **button_to_remote**. En lugar de crear un formulario AJAX, se agregan botones asíncronos

L29 Descarga de archivos en Rails

A continuación se muestra el desarrollo de una aplicación en Rails permitiendo realizar la descarga de archivos.

1. Se crea la aplicacion upload
   $ rails new upload
   Se cambia al directorio upload
2. Se crea el modelo, aunque no se requiere base de datos se llamara DataFile
   $ rails generate model Datafile
3. En el archive datafile.rb se agrega el método save, que será llamado por el controlador.

```
class DataFile < ActiveRecord::Base

  def self.save(upload)

    name =  upload['datafile'].original_filename

    directory = "public/data"

    # create the file path

    path = File.join(directory, name)

    # write the file

    File.open(path, "wb") { |f| f.write(upload['datafile'].read) }

  end

end
```

El metodo definido tomar el objeto upload CGI y extraerá el nombre del archivo y lo almacenar en el directorio "public/data" directory. Tambien se puede emplear la función content_type para conocer el tipo de medio del arhivo descargado.

File es un objeto ruby y join un método para concatenar el directorio con el nombre del archivo.

4. Crear el controlador Upload
   $ rails generate controller Upload

Ahora se crearan dos funciones de controlador. La primera función index invocara a un archivo de vista para tomar la entrada del usuario y la segunda función uploadFile toma la información del usuario y se la pasa al modelo de 'DataFile'. Se inicializa el directorio de 'uploads' como se creó anteriormente "directory = 'data'".

```
class UploadController < ApplicationController

  def index

    render :file => 'app\views\upload\uploadfile.rhtml'
```

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

```
  end

  def uploadFile

    post = DataFile.save(params[:upload])

    render :text => "File has been uploaded successfully"

  end

end
```

5. Crear la vista:

Finalmente se crea el archivo uploadfile.rhtml con el código siguiente:

```
<h1>File Upload</h1>

<%= start_form_tag ({:action => 'uploadFile'},

                    :multipart => true) %>

<p><label for="upload_file">Select File</label> :

<%= file_field 'upload', 'datafile' %></p>

<%= submit_tag "Upload" %>

<%= end_form_tag %>
```

L30 Envío de emails en Rails

Action Mailer es el componente de Rails que habilita a las aplicaciones para enviar y recibir emails.

A continuación se muestra el desarrollo de una aplicación en Rails permitiendo realizar la descarga de archivos.

1. Se crea la aplicación emails
   $ rails new emails
   Se cambia al directorio emails
2. Se configura Action Mailer.
   Ir a archivo de configuración **environment.rb** y agregar la siguiente línea al final del archivo.

```
ActionMailer::Base.delivery_method = :smtp
```

Esta acción le indica a ActionMailer se va a utilizar un servidor SMTP, también puede especificarse sendmail.

Se agregan también las siguientes líneas al archivo environment.rb.

```
ActionMailer::Base.server_settings = {

  :address => "smtp.correo.com",

  :port => 25,
```

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

```
  :domain => "correo.com",
  :authentication => :login,
  :user_name => "usuario",
  :password => "contrasena",
}
```

Se deben colocar los valores reales del servidor de correo.

Tambien se pueden definir los formatos de correo agregando la siguiente línea al archivo environment.rb

ActionMailer::Base.default_content_type = "text/html"

ActionMailer::Base.default_content_type puede ser parametrizada en "text/plain", "text/html", y "text/enriched". El valor defecto es "text/plain".

3. Se crea un mailer emailer
   $ rails generate mailer emailer

Esto creara un archivo emailer.rb en el directorio app\mailers:

```
class Emailer < ActionMailer::Base
 default from: "from@example.com"
end
```

4. Se crea un método contact de la siguiente manera

```
class Emailer < ActionMailer::Base
  def contact(recipient, subject, message, sent_at = Time.now)
    @subject = subject
    @recipients = recipient
    @from = 'no-reply@yourdomain.com'
    @sent_on = sent_at
          @body["title"] = 'This is title'
          @body["email"] = 'sender@yourdomain.com'
          @body["message"] = message
    @headers = {}
  end
end
```

5. Crear un archivo en la vista

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

Colocar el siguiente código en el archivo app/views/emailer/contact.rhtml file

```
Hi!

You are having one email message from <%= @email %> with a title

<%= @title %>

and following is the message:

<%= @message %>


Thanks
```

6. Luego se genera el controlador

```
$ rails generate controller emailer
```

Se debe definir un método en el controlador emailer_controller.rb el cual llama al método del modelo:

```
class EmailerController < ApplicationController
  def sendmail
    email = @params["email"]
            recipient = email["recipient"]
            subject = email["subject"]
            message = email["message"]
    Emailer.deliver_contact(recipient, subject, message)
    return if request.xhr?
    render :text => 'Message sent successfully'
  end
end
```

Para entregar el e-mail utilizando el método contact debe agregarse **deliver_** al inicio del nombre del método. La función request.xhr? permite enviar un mensaje de texto si el navegador no soporta javascript.

7. Finalmente se prepara un formulario donde se toma la información del usuario para enviar el email.

Agregar el siguiente código al archivo in emailer_controller.rb

```
  def index
    render :file => 'app\views\emailer\index.rhtml'
  end
```

Se define la vista en app\views\emailer\index.rhtml

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela

```
<h1>Send Email</h1>

<%= start_form_tag :action => 'sendmail' %>

<p><label for="email_subject">Subject</label>:

<%= text_field 'email', 'subject' %></p>

<p><label for="email_recipient">Recipient</label>:

<%= text_field 'email', 'recipient' %></p>

<p><label for="email_message">Message</label><br/>

<%= text_area 'email', 'message' %></p>

<%= submit_tag "Send" %>

<%= end_form_tag %>
```

Todos los derechos reservados, prohibida la reproducción, Instituto Gala de Venezuela